
**Instituto Superior de Engenharia de Lisboa**

[redacted]

**Redes de Computadores**

[redacted]

**Trabalho Prático 1/Fase 1**

**Docente Luis Pires**

**Realizado por:**

[redacted]

[redacted]

# Índice Geral

# Índice de Figuras

# 1. Introdução.

Nesta 1º fase do projeto foram propostos os seguintes objetivos:

- Instalação e configuração de um web server, tendo sido usada a distribuição XAMPP do Apache.
- Testes de acesso ao web server a partir de outro dispositivo da rede.
- Testes e monitorização de pedidos ao servidor e respetivas respostas, comparando os headers de ambas usando o Wireshark.
- Desenvolvimento de um cliente web para fazer pedidos ao servidor web instalado e a outros servidores na internet.

Além dos objetivos propostos, e numa lógica de compreensão mais aprofundada do funcionamento dos servidores e clientes web, foram acrescentados os seguintes objetivos:

- Criação de uma estrutura no servidor capaz de simular algumas respostas e respetivo tratamento dessas respostas do lado do cliente.
- Criação de um modelo de dados para tratamento das respostas do servidor e dos pedidos do cliente.
- Simulação de uma situação de cache por parte do cliente, implementando uma lista de endereços visitados e datas de modificação das respetivas páginas.
- Criação de uma implementação o mais robusta possível, criando rotinas de teste aos nomes dos servidores introduzidos (hostnames) e fazendo o tratamento dos inputs do utilizador nos menus da aplicação, evitando na medida do possível o aparecimento de exceções que provocassem a quebra da aplicação.
- Criação de rotinas para possibilitar a leitura de todo o pacote transmitido pelo servidor e não apenas os primeiros 1024 bytes.

## 2. Instalação e configuração do WebServer.

O WebServer utilizado foi o Apache com base na distribuição XAMPP. Esta distribuição cria uma máquina virtual (VM) Linux (Debian, figura 1) na máquina host, uma rede privada 192.168.64.0/24, uma interface (bridge) com o ip 192.168.64.1 na máquina host, ficando a VM com o ip 192.168.64.2. Uma stack de serviços (Apache, MariaDB e ProFTPD) é criada na VM no ip 192.168.64.2. É possível fazer reencaminhamento de pacotes (figura 2) entre a máquina host e a VM, para que seja possível encaminhar para os serviços da stack, pedidos que chegam às portas da máquina host.
No caso do MacOS, quando a configuração de reencaminhamento foi feita, foi criada uma regra na firewall para permitir a entrada dos pacotes pela porta 80 (figura 3).

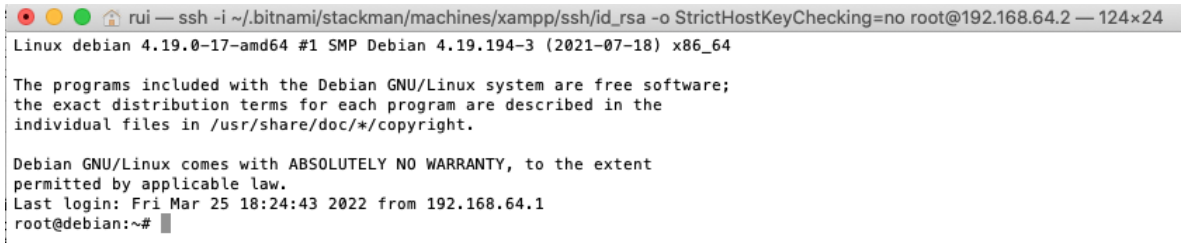


*Figura 1 - Shell da VM Debian no terminal do MacOS.*

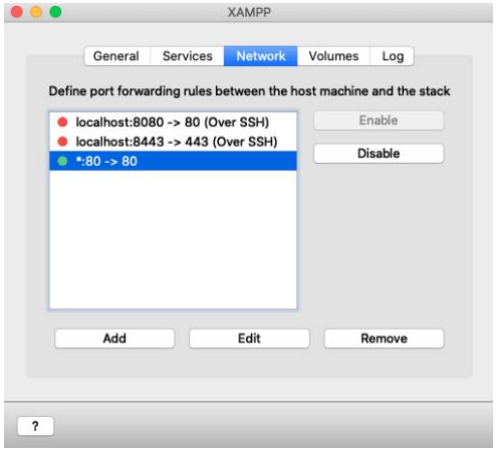


*Figura 2 - Janela de confFigura 3 - Regra da firewall do MacOS que permite o reencaminhamento de pacotes.*

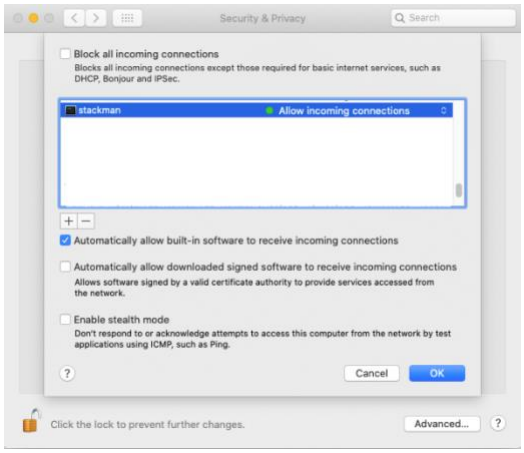


*Figura 4 - Regra da firewall do MacOS que permite o reencaminhamento de pacotes.*

## 3. Testes ao WebServer.

Foram realizados testes de acesso ao servidor web a partir de um iPad. As figuras 4,5,6,7 e 8 apresentam as capturas do Wiresharck resultantes desse teste.

**Figura 5** - Foi enviado um pedido para o host 192.168.8.5 com o endereço "/" (raiz do site). Podemos verificar que o browser envia um campo "Connection: Keep-alive" solicitando que o servidor não termine imediatamente a sessão.

**Figura 6** - O servidor respondeu com um código de status 302 Found, que indica que a página se encontra noutra localização. Esta não é uma nova localização permanente, nesse caso o código seria o 301 Moved. Verifica-se que o servidor aceita o Keep-alive e fornece a informação "Keep-Alive: timeout=5, max=100", indicando o período máximo em segundos que a sessão pode permanecer ativa (5 segundos) e o número máximo de pedidos que podem ser enviados na sessão (100).

**Figura 7** – O browser envia novo pedido de acesso com a nova localização (/dashboard/).

**Figura 8** – O servidor responde com um status "200 OK" indicando que o pedido foi atendido. Note-se também o decremento do máximo de pedidos possíveis na sessão para 99, visível no campo "KeepAlive".

Em relação ao KeepAlive, a literatura lida sobre o assunto refere que o parâmetro "max" deste campo pode ser usado por um serviço HTTP pipeline para limitar a quantidade de pedidos no pipeline. Refere-se também que valores do parâmetro "timeout" maiores que o estipulado pela camada de transporte são anulados (fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Keep-Alive).

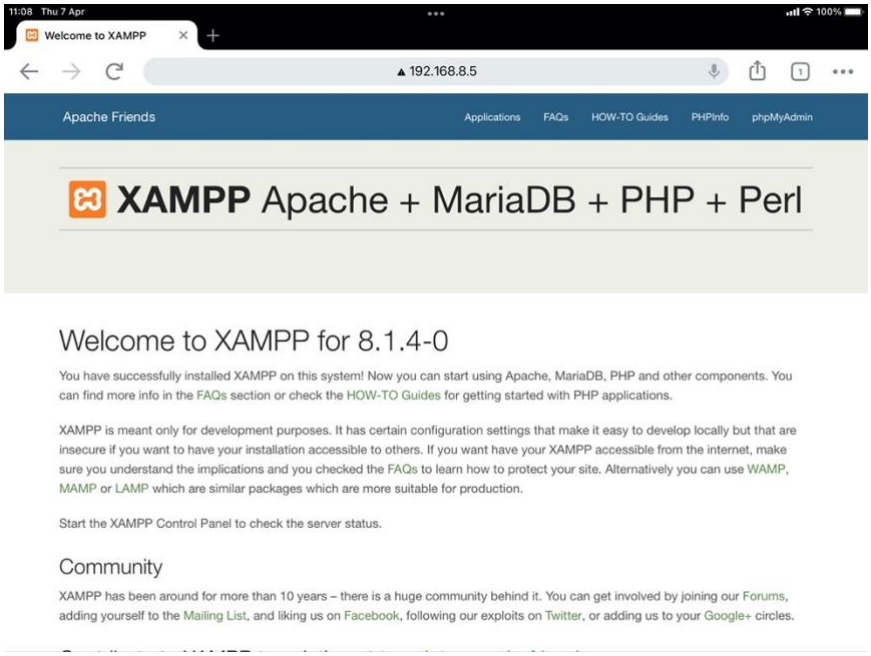


*Figura 5 - Resultado do acesso à página inicial do Apache a partir de um iPad.*

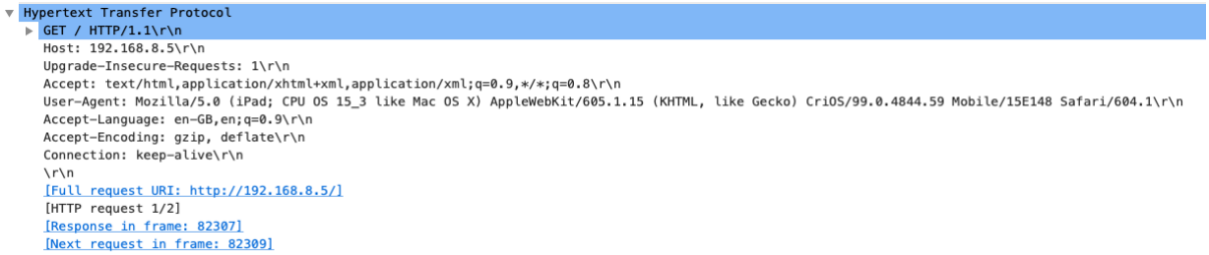


*Figura 6 - Captura do Wiresharck do pedido do Safari (cliente) ao endereço "/" do servidor Apache.*

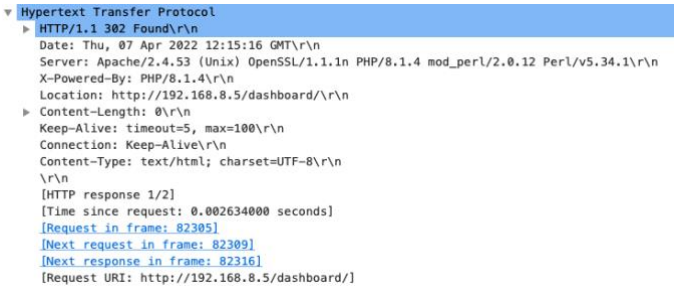


*Figura 7 – Captura do Wiresharck da resposta ao pedido anterior.*

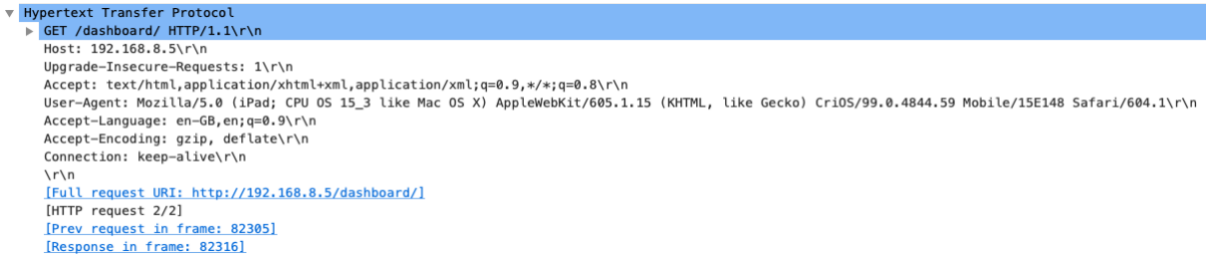


*Figura 8 – Captura do Wiresharck do novo pedido do Safari contendo a nova localização indicada na resposta anterior.*

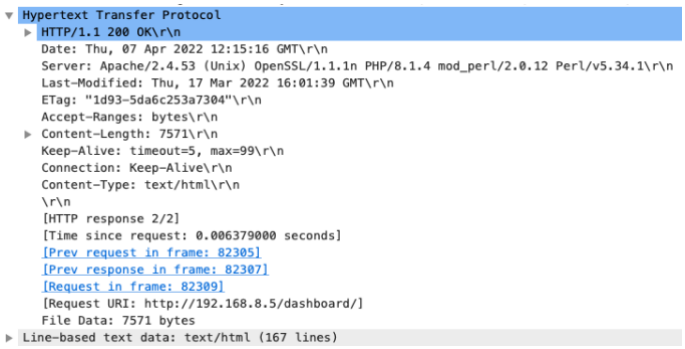


*Figura 9 - Captura do Wiresharck da resposta do servidor Apache ao pedido anterior do Apache.*

## 4. Estrutura de pastas e ficheiros no servidor Apache.

Foi criada uma estrutura de pastas para fazer os testes no servidor. A pasta htdocs/rc deve ser copiada para a pasta htdocs do servidor Apache.

**/rc/** - Raiz do site do trabalho e também para fazer pedidos que retornam um status 200 OK. Contém um ficheiro index.html com o seguinte código:

```
<!DOCTYPE html>
<html>
<body>
<p>Hello World!!!</p>
</body>
</html>
```

**/rc/redirect/** - serve para fazer pedidos que retornam um status 301 Moved Permanently. Contém um ficheiro .htaccess com o seguinte código:

```
Redirect 301 /rc/redirect/ /rc/newlocation/
```

**/rc/newlocation/** - serve como destino para o reencaminhamento feito na pasta /rc/redirect/. Contém um ficheiro index.html com o seguinte código:

```
<!DOCTYPE html>
<html>
<body>
<p>Redirect OK</p>
</body>
</html>
```

**/rc/unauthorized/** - serve para fazer pedidos que retornam um status 401 Unauthorized. Contém um ficheiro .htaccess com o seguinte código:

```
AuthType Basic
AuthName "Restricted Resource"
Require group admin
```

**/rc/forbidden/** - serve para fazer pedidos que retornam um status 403 Forbidden. Contém um ficheiro .htaccess com o seguinte código:

```
Require all denied
```

# 5. O cliente HTTP.

## 5.1 Funcionalidades do cliente.

O cliente HTTP possui uma interface de texto simples e intuitiva que corre numa linha de comandos respondendo os inputs passados pelo utilizador.


*Figura 10 - Interface do utilizador – Menu principal*

No topo do menu principal é descrito o nome do host, que por defeito será o "localhost". É possível passar o hostname como argumento na linha de comandos, usando a seguinte forma:

#>python interface.py <hostname>

Exemplo:

#> python interface.py www.google.com

De seguida são apresentados três parâmetros:

- **Print header: on** – Estado da exibição dos dados do header.
- **Print data: on** – Estado da exibição dos dados do payload.
- **Fake cache entries: #** - indica o número entradas na lista (dicionário) de requests realizados pelo utilizador cuja resposta do servidor tinha status "200 OK" e que continha o campo "Last-Modified". Este dicionário guarda como chave o conjunto <host,endereço> e como valor a data do campo "Last-Modified". Desta forma simula-se uma cache, podendo os endereços desta lista ser comparados com os endereços de outros pedidos, e se forem iguais, enviar o campo "If-Modified-Since" com a data do valor do dicionário.

O menu principal tem as seguintes opções:
**0: Quit** – Sair da aplicação.
**1: Change host** – Mudar o hostname do servidor http atual.
**2: Send GET** – Enviar um pedido com o método GET para um endereço no servidor definido pelo hostname.
**3: Print header on/off** – Ligar e desligar a exibição dos dados do header das mensagens de resposta.
**5: Print data on/off** – Ligar e desligar a exibição dos dados do payload das mensagens de resposta.
**6: Show fake cache** – Mostra o conteúdo da lista da cache simulada.
**9: Requests menu for localhost testes…** - Menu com pedidos pré-definidos ao servidor Apache instalado na máquina local (localhost).

Ao escolher a opção 9 do menu principal, o utilizador acede ao “REQUESTS MENU”. Neste menu existem pedidos pre-definidos que provocam respostas com alguns status possíveis, entre eles status com os erros mais comuns.

```
================== REQUESTS MENU ==================
0: Return to main menu...
1: Server returns a 200 OK status code the first time. After that returns a 304 Not Modified. (address sended: /rc/)
2: Server returns a REDIRECTION status (address sended: /rc/redirect/)
3: Server returns a 400 Bad Request status error (address sended: dummy)
4: Server returns a 401 Unauthorized status error (address sended: /rc/unauthorized/)
5: Server returns a 403 Forbidden status error (address sended: /rc/forbidden/)
6: Server returns a 404 Not found status error (address sended: /rc/dummy/)
---------------------------------------------------
Choose option (0):
```

*Figura 11 - URLs de teste HTTP*

Cada opção apresentada na imagem anterior faz um request para a URL “address sended”, obtendo o código e mensagem padrão HTTP.

OS dados das respostas do servidor, são apresentados da seguinte forma:

- pedido HTTP enviado (RESQUEST SENDED).
- status detalhado (STATUS).
- o cabeçalho completo com todos os campos ().
- o payload deste pedido.

Figura seguinte, podemos observar o aspeto da resposta do servidor Apache a um pedido feito através da opção 1 do menu “REQUESTS MENU”.

```
Choose option (0):1

+++++++++ REQUEST SENDED ++++++++++
GET /rc/ HTTP/1.1\r\n
Host: localhost\r\n
\r\n

+++++++++ RESPONSE BEGIN ++++++++++
++++++++++++++ STATUS ++++++++++++++
Status code: 200
Status reason: OK
Status type: StatusType.SUCCESS
Status info: The request was fulfilled.
++++++++++++++ HEADER ++++++++++++++
HTTP/1.1 200 OK
Date: Sat, 09 Apr 2022 19:51:58 GMT
Server: Apache/2.4.53 (Unix) OpenSSL/1.1.1n PHP/8.1.4 mod_perl/2.0.12 Perl/v5.34.1
Last-Modified: Tue, 05 Apr 2022 12:55:42 GMT
ETag: "4c-5dbe7c33230d0"
Accept-Ranges: bytes
Content-Length: 76
Content-Type: text/html

++++++++++++++ PAYLOAD ++++++++++++++
<!DOCTYPE html>
<html>
<body>

<p>Hello World!!!</p>

</body>
</html>

++++++++++ RESPONSE END ++++++++++
```

*Figura 12 - Resposta HTTP ao servidor local*

O objecto StatusCode contém um atributo “info” onde está guardada a explicação do status.

### 5.1.1 Pedidos HTTP a servidores web na internet.

De seguida será exemplificado um pedido feito a um servidor externo, o Google, onde iremos obter a homepage do google images. Primeiro mudamos o host para o destino desejado usando a opção 1 do menu principal:

```
Choose option (9):1
Hostname?: www.google.com

Current server hostname: www.google.com
Print header: on | Print data: on  | Fake cache entries: 0
======================= MAIN MENU ========================
0: Quit
1: Change host
2: Send GET
3: Print header on/off
4: Print data on/off
5: Clean fake cache
6: Show fake cache
9: Requests menu for localhost testes...
----------------------------------------------------------
```

*Figura 13 – Mudando o hostname para www.google.com.*

Depois é feito um pedido para a URL do google images (/images) usando a opção 2 do menu principal. A resposta é uma página de redirecionamento para o novo endereço apresentado pela entidade.

```
Choose option (9):2
Address?:/images

+++++++++ REQUEST SENDED ++++++++++
GET /images HTTP/1.1\r\n
Host: www.google.com\r\n
\r\n

++++++++ RESPONSE BEGIN +++++++++
++++++++++++ STATUS ++++++++++++++
Status code: 302
Status reason: Found
Status type: StatusType.REDIRECTION
Status info: None
++++++++++++++ HEADER +++++++++++++
HTTP/1.1 302 Found
Location: http://www.google.com/imghp
Cache-Control: private
Content-Type: text/html; charset=UTF-8
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Date: Fri, 08 Apr 2022 18:43:25 GMT
Server: gws
Content-Length: 224
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
```

*Figura 14 - Primeira resposta do request Google Images*

Como podemos observar na figura anterior, obteve-se um código “302 Found” e também um campo “Location” com o endereço atual do serviço.

O cliente, ao receber a resposta, questiona o utilizador se ele está interessado em prosseguir o redirecionamento do pedido ou terminá-lo.

```
We received a 302 Found status. The new location is http://www.google.com/imghp
Press enter to request the new location, press q followed by enter to quit:
```

*Figura 15 - Pedido de redirecionamento feito pelo utilizador no Cliente*

Caso o utilizador decida prosseguir, é feito o novo pedido para a URL fornecida pelo servidor. Nas figuras seguintes, podemos ver esse pedido de reencaminhamento, o status da respostas, bem como o restante header e o payload.

```
+++++++++ REQUEST SENDED ++++++++++
GET http://www.google.com/imghp HTTP/1.1\r\n
Host: www.google.com\r\n
\r\n
```

*Figura 16 - Segundo pedido para o endereço fornecido pelo serviço*

```
+++++++++ RESPONSE BEGIN ++++++++++
++++++++++++ STATUS +++++++++++++++
Status code: 200
Status reason: OK
Status type: StatusType.SUCCESS
Status info: The request was fulfilled.
```

*Figura 17 - Resposta HTTP confirmando o sucesso da resposta*

```
++++++++++++++ HEADER ++++++++++++++
HTTP/1.1 200 OK
Date: Sat, 09 Apr 2022 13:06:23 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
```

*Figura 18 - Header da segunda resposta*

```
++++++++++++ PAYLOAD ++++++++++++
5884
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="pt-PT"><
head><meta content="Google Imagens. A pesquisa de imagens mais abrangente na Web." na
me="description"><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><
meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop=
"image"><title>Google Imagens</title><script nonce="SpCXZhkNnSDFFL64E9i+yQ==">(functi
on(){window.google={kEI:'z4RRYu-OE6GUwbkP5I6I8A8',kEXPI:'0,202454,3,1100077,56875,605
9,206,4804,2316,383,246,5,1354,4013,1238,1122515,1197762,639,380089,16115,28684,17572
```

*Figura 19 - Parte do payload da segunda resposta*

De seguida é apresentada uma sequência de figuras, que mostram o que foi capturado pelo Wireshark, durante os pedidos e respostas do nosso Cliente HTTP.

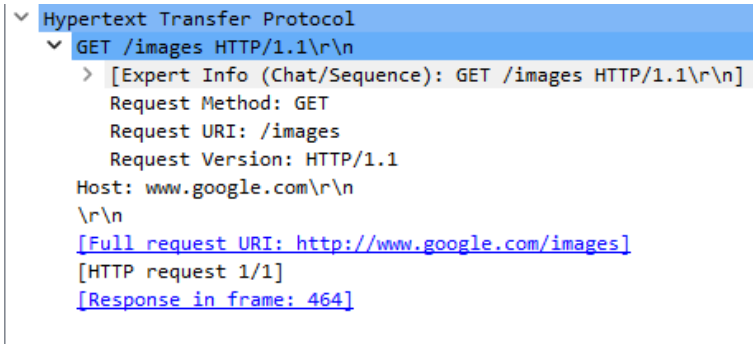


*Figura 20 - Primeiro pedido capturado pelo Wireshark*

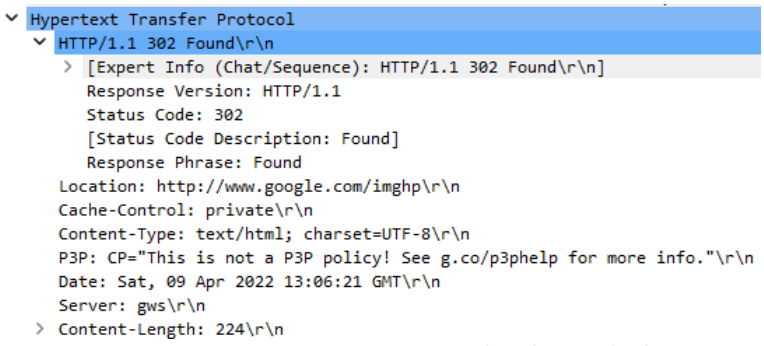


*Figura 21 - Primeira resposta capturada Pelo Wireshark*


*Figura 22 - Segundo pedido HTTP capturado pelo Wireshark*

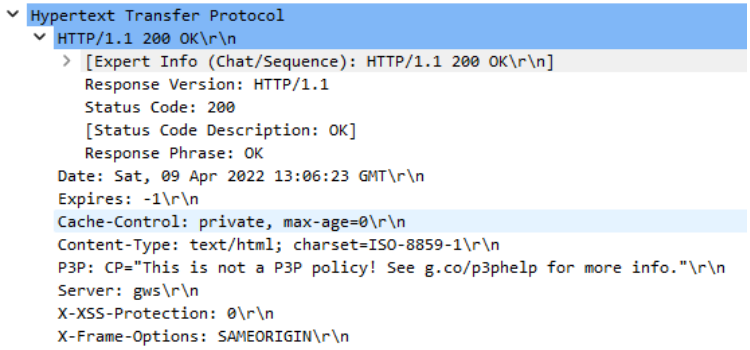


*Figura 23 - Segunda resposta capturada pelo Wireshark*

## 6. Diagrama de classes da aplicação (UML)

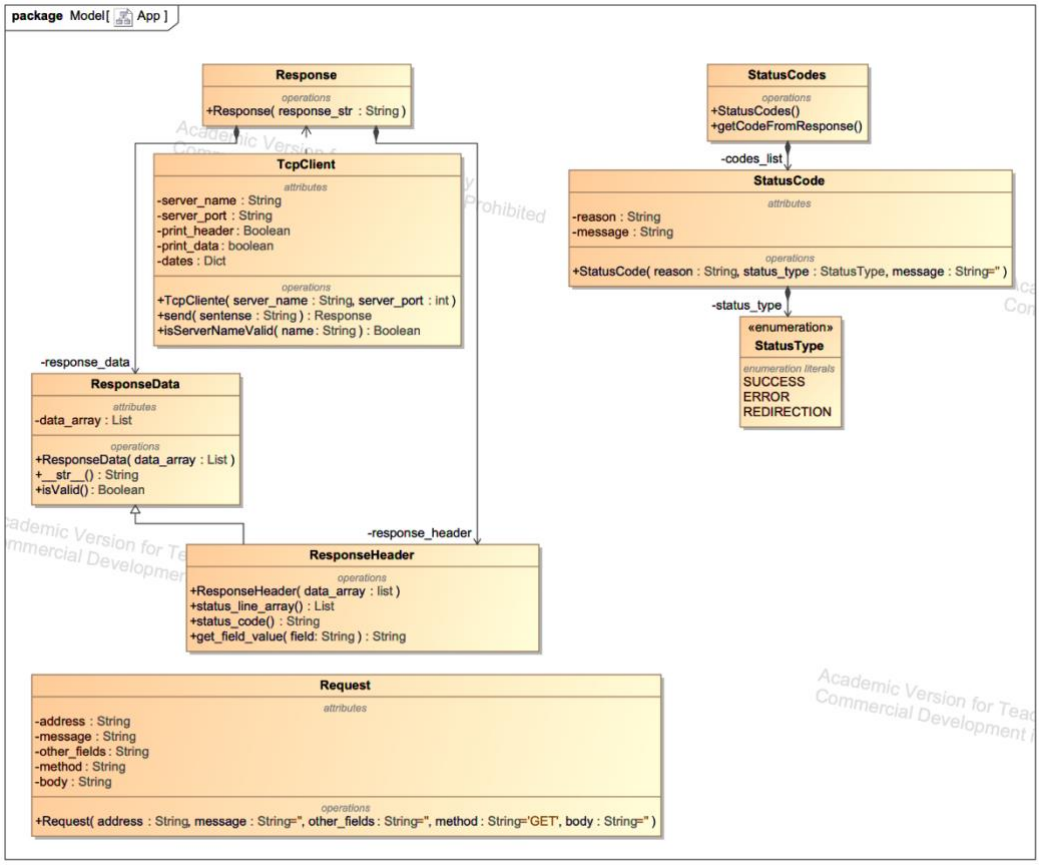


*Figura 24 - Diagrama de classes (UML)*

## 7. Análise crítica

No processo de desenvolvimento do Cliente HTTP, nomeadamente na classe "TcpClient", foi implementada uma solução mais elaborada em relação ao recebimento dos dados da resposta HTTP. Num ciclo "while", para o tratamento da resposta, são lidos 1024 bytes e caso seja encontrado o código "304" a aplicação termina o ciclo de leitura, uma vez que não houve alterações na informação obtida anteriormente, caso o contrário, é feito uma leitura do campo "Content-Length" presente na resposta, se for zero ou menor que 1024 bytes o ciclo é encerrado, no entanto, se o valor do payload for superior aos 1024 bytes, este número é usado na próxima iteração, desta maneira, obtém-se uma forma de leitura dos bytes recebidos com um tamanho variável.

O Cliente HTTP, também faz o tratamento dos códigos de erros, que são devolvidos nas respostas geradas pelos servidores, locais ou não, imprimindo o erro que foi retornado na linha de comandos para o utilizador.

Foram implementados alguns ficheiros (.htaccess) de configuração no servidor local, servindo para exemplificar alguns erros típicos e conhecidos do HTTP, como 400 Bad Request, 401 Unauthorized, 403 Forbidden e o clássico 404 No found.

## 8. Conclusão.

O protocolo HTTP é um protocolo que funciona sobre a camada de transporte TCP, sendo essêncialmente um protocolo da camada de aplicação. As mensagens são constituidas por:

- **Header**: contém um campo de status obrigatório e vários outros campos com funções diversas. Cada campo é separado dos outros por uma quebra de linha. Os campos são constituidos por um nome e um valor ou sequências de valores obdecendo a um formato do tipo **nome: <valores>.**
- **Content Data (payload)**: contém o corpo da mensagem, normalmente o conteúdo a ser apresentado no browser. Este parte da mensagem é separada do header por uma quebra de linha sem qualquer outra informação. O header pode ou não conter um campo **Content-Length** que indica o tamanho do content-data. Outra forma de indicar o limite da content-data é através do campo Content-Type e do seu valor **boundary** que atua como delimitador do coteúdo da mensagem.

Durante esta fase do trabalho, foram estudados alguns aspetos destes dois componentes da mensagem HTTP, e desenvolvido um cliente com algoritmos para tratar uma pequena parte da informação contida nestes elementos, principalmente no header da mensagem.

## 9. Bibliografia.


- https://www.w3.org/Protocols/HTTP/HTTP2.html
- https://developer.mozilla.org/pt-BR/docs/Web/HTTP